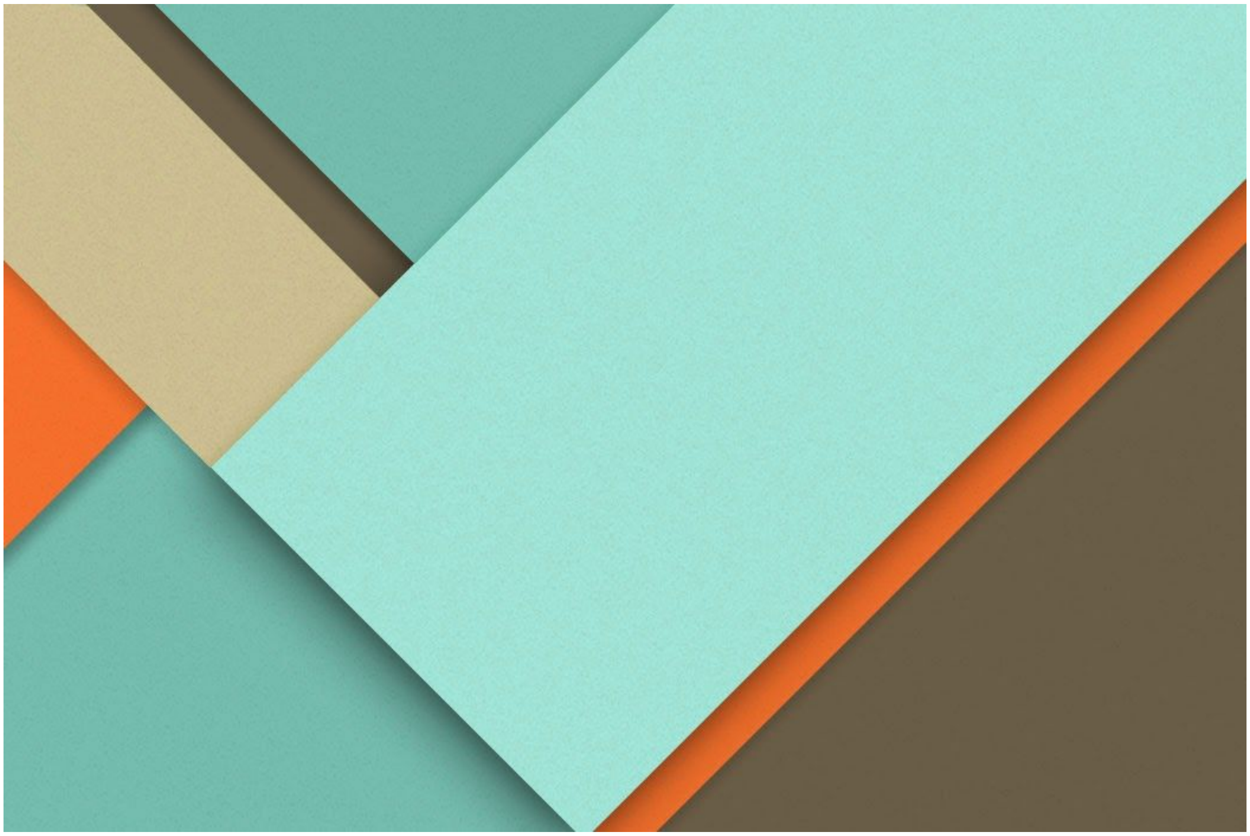


Car price prediction report

06.29.2020

Group members: Pouya kazemian, Mohammad yousefi, majid delqandy
University Of Guilan
Professor's name: Professor Tourani

Overview

We are about to build an Artificial Neural Network which can predict the price of a given vehicle by the information related to it (the model, engine capacity, etc...).

First of all, the dataset we have, has to be cleansed (removing unrelated information or NaN values or filling the NaN values, etc...) and needs some data preprocessing.

After that, we need to build a model (ANN) and feed a fraction of the dataset, and then we test it with the other fraction to see whether the model can predict the prices with a good precision or not.

Goals

1. Cleaning the data correctly and not leaving any loose ends behind (Data leakage, ...).
2. Feeding the model with the minimum epochs without underfitting or overfitting it.
3. Getting good accuracy and upgrading it as much as we are able to.

Specifications

1. Data Cleaning and Preprocessing

As you can see in the picture below, we are using some libraries for our work first; After that, we are reading the CSV file that contains the data we need.

After reading the dataset, some data preprocessing is happening; Some columns are being removed which have unrelated information, some rows containing NaN values are being dropped and the remaining columns containing numbers are being converted to float type.

```python
import pathlib

import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import seaborn as sns

import tensorflow as tf

from tensorflow import keras
from tensorflow.keras import layers
from keras import models
```

```python
df = pd.read_csv('/content/drive/My Drive/Colab Notebooks/CARS.csv')
df = df.drop(['Model','DriveTrain','Invoice','Origin','Type'], axis=1)
df = df.dropna()
df.head()
```

	Make	MSRP	EngineSize	Cylinders	Horsepower	MPG_City	MPG_Highway	Weight	Wheelbase	Length
0	Acura	$36,945	3.5	6.0	265	17	23	4451	106	189
1	Acura	$23,820	2.0	4.0	200	24	31	2778	101	172
2	Acura	$26,990	2.4	4.0	200	22	29	3230	105	183
3	Acura	$33,195	3.2	6.0	270	20	28	3575	108	186
4	Acura	$43,755	3.5	6.0	225	18	24	3880	115	197

```python
df.isna().sum()
df['MSRP'] = df['MSRP'].str.replace('$','').str.replace(',','').astype(float)
df['Horsepower'] = df['Horsepower'].astype(float)
df['MPG_City'] = df['MPG_City'].astype(float)
df['MPG_Highway'] = df['MPG_Highway'].astype(float)
df['Weight'] = df['Weight'].astype(float)
df['Wheelbase'] = df['Wheelbase'].astype(float)
df['Length'] = df['Length'].astype(float)
df.dtypes
```

```
Make            object
MSRP           float64
EngineSize     float64
Cylinders      float64
Horsepower     float64
MPG_City       float64
MPG_Highway    float64
Weight         float64
Wheelbase      float64
Length         float64
dtype: object
```

.

After that, we are mapping the vehicle to 0 and 1 by their model with get_dummies function;

For example if a vehicle is an audi, the audi column in this row will be 1 and the other column models will be 0.

For normalizing our data, we used 'RobustScaler' because it was suggested for reducing the influence of the outliers in the dataframe. Its effect was phenomenal; before using this scaler the EPOCHS was 5000, now it's only 800 which is much faster.

2. Building the model and training process

Our model has 4 layers: 1 input layer with 64 neurons, 2 hidden layers and 1 output layer(picture below) and their activation function is 'relu' which is widely used.

For metrics we've used mean_absolute_error(mae) and mean_squared_error(mse); and now we compiled the model and you can see a summary of the model below.

```
[132] from sklearn.preprocessing import RobustScaler

      scaler = RobustScaler()
      train_df = scaler.fit(train_df).transform(train_df)
      test_df = scaler.fit(test_df).transform(test_df)
```

```
[145] model = keras.Sequential([
          layers.Dense(64, activation='relu', input_shape=[46]),
          layers.Dense(32, activation='relu'),
          layers.Dense(8 , activation='relu'),
          layers.Dense(1)
        ])
```

```
[146] optimizer = tf.keras.optimizers.RMSprop(0.001)
      model.compile(loss='mse', optimizer=optimizer, metrics=['mae', 'mse'])
      model.summary()
```

```
Model: "sequential_14"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_63 (Dense)             (None, 64)                3008
_________________________________________________________________
dense_64 (Dense)             (None, 32)                2080
_________________________________________________________________
dense_65 (Dense)             (None, 8)                 264
_________________________________________________________________
dense_66 (Dense)             (None, 1)                 9
=================================================================
Total params: 5,361
Trainable params: 5,361
Non-trainable params: 0
_________________________________________________________________
```

```
[147] EPOCHS = 800
      history = model.fit(train_df, train_label, shuffle=True, epochs=EPOCHS, validation_split = 0.2, verbose=0)
      hist = pd.DataFrame(history.history)
      hist['epoch'] = history.epoch
      hist.tail()
```

	loss	mae	mse	val_loss	val_mae	val_mse	epoch
795	58857452.0	4436.207031	58857452.0	80940424.0	6243.264648	80940424.0	795
796	58930552.0	4459.094238	58930552.0	80975800.0	6239.711426	80975800.0	796
797	58854576.0	4441.794434	58854576.0	80998096.0	6236.111816	80998096.0	797
798	58975544.0	4427.550293	58975544.0	80943136.0	6246.484375	80943136.0	798
799	58903264.0	4426.543945	58903264.0	81023440.0	6256.218750	81023440.0	799

Now we fit the model with the training data which is 80 percent of the entire dataset for 800 rounds(epoch). The result can be seen in the picture above.

3. Evaluating and testing process

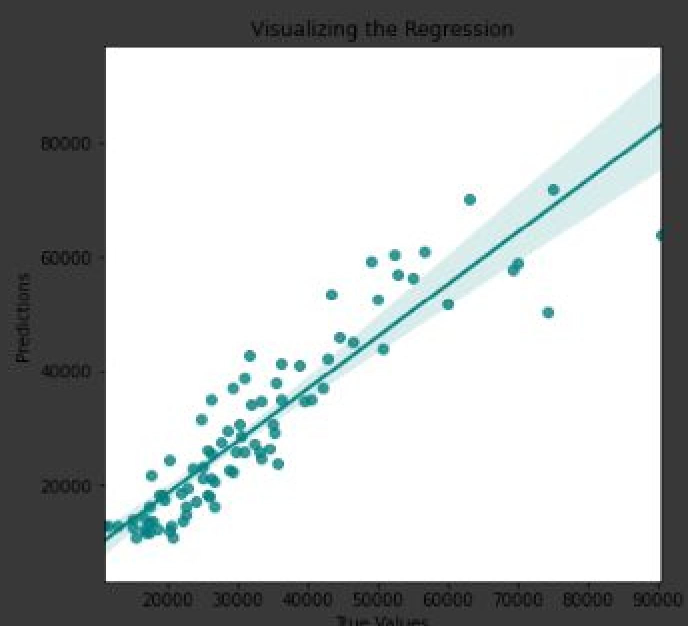
Now we evaluate the metrics loss, mae and mse. After that, we predict the test_set data and plot it on a regression plot. The nearer the dots are to the line, the better the accuracy will be and lower the metrics will be. The r2_score is almost 81.4% which is pretty good.

```
[ ] loss, mae, mse = model.evaluate(test_df, test_label, verbose=2)
```

```
3/3 - 0s - loss: 47709204.0000 - mae: 5306.1665 - mse: 47709204.0000
```

```
[ ] test_predictions = model.predict(test_df).flatten()

    plt.figure(figsize= (6, 6))
    plt.title('Visualizing the Regression')
    sns.regplot(test_label, test_predictions, color = 'teal')
    plt.xlabel('True Values')
    plt.ylabel('Predictions')
    plt.show()
```




```
[ ] from sklearn.metrics import r2_score
    print('The R2 square value of NN is :', r2_score(test_label, test_predictions)*100)
```

```
The R2 square value of NN is : 81.36412669346828
```